AD-A230 656

DTIC FILE COPY

DTIC ELECTE FEB 01 1991 S D D

EAVE-West: A Testbed For Plan Execution

Jayson T. Durham, Paul Heckman, Dale Bryan, and Ron Reich

Undersea Artificial Intelligence and Robotics Branch
Ocean Engineering Division
Naval Ocean Systems Center
San Diego, CA 92152-5000

1987

Abstract

The Experimental Autonomous Vehicle - West (EAVE-West) submersible testbed has been configured for demonstrating a distributable software architecture for Autonomous Undersea Vehicle (AUV) plan execution. Instead of using a machine planner aboard the AUV, plans are represented and then downloaded to the vehicle. This technique obviates the problems associated with planning and, as a result, the real-time response of the AUV can potentially be improved. A review of the architecture is given and the EAVE-West demonstration system is discussed. Keywords: Underwater vehicles; Submersibles test beds; Distributed data processing; Artificial intelligence. (EDC)

Introduction

To potentially reduce costs, reduce personnel risks, and increase overall system performance, undersea vehicle tasks are being automated. Remotely Operated Vehicles (ROVs) have already exploited a certain limited automation capability through the use of embeddable microprocessors, real-time software, and Artificial Intelligence techniques [Doeling and Harding 87]. In particular, supervisory controlled or "telerobotic" ROVs have progressively automated vehicle control tasks which previously required the attention of a human operator. By developing structured methodologies that will accelerate and advance the degree of ROV task automation, more flexible autonomous capability can be made possible for future vehicle systems.

Underwater ROVs developed from the need to perform such generic missions as undersea search, recovery, and inspection operations. Originally, such unmanned undersea vehicles were basically teleoperators or controlled via a master-slave configuration, and therefore, none of the vehicle tasks were automated. Dr. Tom Sheridan developed a system architecture he called supervisory control, whereafter NOSC demonstrated this supervisory control architecture on an undersea manipulator [Yoerger and Sheridan 83]. This means of control was adopted for developing advanced untethered submersibles, as well. Using this system architecture, tasks previously performed manually, could be automated. Because a microprocessor could be embedded into a vehicle, a vehicle could be configured to perform well defined

91 1 30 080

Approved for public release;
distribution unlimited.

tasks on command. An operator can issue high level commands, and thereafter supervise the execution of those commands. Due to the development of supervisory control, advanced undersea vehicles have became a reality.

Concurrent with the development of supervisory control, there has been speculation on the possibility of fully automated ROV systems. This speculation has led to the concept of Autonomous Undersea Vehicles (AUVs). Historically, machine intelligence and problem solving have been emphasized at the cost of specific mission capability. In contrast to this historical trend, an autonomous vehicle is herein considered to be an advanced supervisory controlled vehicle which has been extended with mission level command capability. It is not necessarily "intelligent." Figure 1 illustrates this continuum of ROV capabilities. Autonomy is defined here as meaning that the vehicle can be commanded to automatically perform complete mission tasks. This approach incorporates the successes of previous vehicle developments, such as supervisory control, and addresses autonomoy as a more realistically attainable advanced automation problem than a problem of "Artificial Intelligence" (AI). This conceptually offers an approach which can provide deterministic solutions and known actions against unpredictable environmental conditions and hardware failure.

Computer technology is the basis of current automation trends. With the development of general purpose computers and resultant computer programming languages, large software systems have automated relatively sophisticated processes. Because software development is a new endeavor, new problems have emerged. Large software projects typically cost much more than expected, usually take much longer to complete, and the final systems are often unreliable.

To control these software problems, software engineering methods have been defined for constructing automated systems. Some of these methods should be applied and adapted to the automation of ROV missions. For this approach, there are two basic challenges. Building mission level commands is an effort for which there are currently no formalized methods. Secondly, mechanisms for executing mission level commands are yet to be developed. This effort has approached these two challenges by developing a methodology for creating and executing mission level tasks, making possible the autonomous execution of ROV missions.

Decision and planning aids can assist with mission planning. These aids are "tools" which are useful for determining mission feasibility and for determining how to achieve a mission objective given the available resources. These software tools assist in the creation of detailed mission plans. Once a detailed mission plan is created, it can be configured into a mission level command.

Declarative programming languages allow a programmer to declare what needs to be done and then the computer is left to

DTIC COPY INSPECTED

es

Dist | Avail and/or Special
A-1 | |

determine how to actually perform the desired operation. Declarative languages allow the programmer to focus his attention on what needs to be implemented. The computer is left to determine how the desired operation is to be executed. A similar objective is desired here, and therefore, a similar approach is adopted. The idea of a plan execution system is to define plan execution primitives such that if a mission plan is represented using those primitives, the computer will already have a mechanism for determining how to execute that mission plan. The vehicle will be able to automatically execute a desired plan.

Designing and demonstrating a plan execution system is the primary objective of this approach toward automated mission execution. An object-oriented software architecture [Booch 83, Booch 86] for mission plan execution has been designed for real-time vehicle control. Mission plans are represented by using two types of abstracted objects: tasks and events. A vehicle plan is represented as an ordered grouping of these task and event objects. Because the system is a distributable software architecture, the vehicle can potentially maintain real-time response independent of the size of the mission plan. Thus, vehicle response becomes directly dependent on the number of processors allocated for the given mission.

Mission Planning Aid

Development and demonstration was attempted for a plan execution system only. Initial tests showed that such an approach was essentially impossible. In order to execute plans, they had to be represented. A represented plan implied that a plan had been created. Creating realistic plans is a problem which was intentionally being avoided. Initial tests showed that the "mission planning" problem had to be addressed.

The development of a Mission Planning Aid (MPA) was consequently initiated for concept demonstration purposes. Given a displayed map, an operator is able to plot a vehicle trajectory. The waypoints of this trajectory are determined by time, depth, and/or heading conditions. The vehicle dead reckons from one point to another. The MPA demonstrated the capability to graphically input and interactively create vehicle trajectory tasks.

At the end of a vehicle trajectory planning session, the MPA creates a file which represents the trajectory plan. This file is transmitted to the plan execution system embedded in the EAVE-West vehicle. The vehicle is then commanded to execute the desired plan. The plan execution system executes the plan.

Mission Plan Execution

Project History

A hierarchical control scheme with conditional execution of tasks was designed for the Ground Surveillance Robot land vehicle

[Harmon and Solorzano 83, Aviles et al 85] and the EAVE-West submersible [Harmon 81, Durham and Shirley 82]. For the NOSC projects, sets of tasks with conditioned initiation and termination were employed for plan execution. Like a production system, conditioned task execution provided a structure which could produce pre-planned, goal-directed behavior. The significant difference between typical production systems and the conditioned task execution was that productions have difficulty representing durations of time and task activities while conditioned tasks are defined in terms of such durations.

The goal of this effort was to define two recursive structures for composing conditioned tasks. Using these structures, an entire mission plan could be represented and then executed without any global control or memory store, such as is provided by a blackboard. Due to this decentralized architecture, a plan execution system built from these structures would be truly extensible. An execution system, which services only these two types of structures, would remain a constant size while plans may be arbitrarily large. Also, if these primitives are configured as augmented trees, the "tree" data structures are well suited for distributed computing machines [Uhr 84]. Using these software structures with distributed computing hardware, a general purpose plan execution system becomes a possibility.

Plan Representation

The Plan Execution System (PES) is defined in terms of Task, Event, and Device objects which are abstractly layered according to their remoteness from the vehicle hardware. Tasks are considered the most abstract objects because they are the most remote from the hardware and they only control the execution of Event objects. Devices are the least abstract, since they interface directly with hardware. Events are a "layer" of objects between Tasks and Devices. Figure 2 shows the layering of these three types of objects.

Task objects provide a procedure abstraction mechanism for creating control hierarchies. Primitive task objects are designed such that "task trees" can be configured in a top-down fashion. A task tree defines a control hierarchy (Figure 3). A set of task trees defines a mission plan. Each tree specifies a task execution interval which is independent of the other task trees within the system. Nodes of a task tree are called control controllers, since they control the interval of execution for sets of subtasks (Figure 4). Subtasks inherit the initiation and termination conditions of their parent tasks. If a task is a leaf within a tree of tasks, its function is to direct the flow of data from the value of an input event to the value of an output event. A leaf task is called a vehicle controller, since its function is to control the vehicle by coupling the value of an input event to the value of an output event (Figure 5).

Tasks are designed to be event couplers. They control the system and do not perform any numerical operations. Tasks are designed to administer vehicle control by coupling input events to output events, given that a start event has occurred and a stop event has not occurred for the given task.

Events provide a data abstraction mechanism for creating hierarchical "polish prefix" expressions. These expressions are binary trees of primitive processes, i.e. objects, which fuse sensor data as well as propagate control values to the vehicle effectors.

A primitive event object has an actual value, which is its current value and it has a desired interval of values. The interval is defined by a minimum desired value and a maximum desired value. A primitive object is said to have a state, as well. An event is initially in the not-occurred state, and remains in the not-occurred state until the actual value is within the desired interval of minimum and maximum values. When the actual value is within the interval of desired values, the event is said to have occurred, and it changes to the occurred state. The event stays in the occurred state until it is reset by its parent object. A node event is simply called a composed event, since it is a binary operation composed of two subevents (Figure 6). A leaf event is called a device event since it is a function of a logical device (Figure 7). A device event can access other event values, event states, or task states. Formally, event values, event states, and task states are considered to be internal device values.

All events have a parent object. The most abstract (i.e., highest level) events are the roots to trees of primitive event objects. These root events are "attached" to parent tasks. They either trigger the start/stop of a task object or they provide input/output for a task object. Events reduce data to control signals which either control the execution of tasks or control the vehicle effectors, e.g. thrusters.

Using the above representation, a mission task is composed of an array of subtasks each with its own start/stop events. A subtask may be a control controller and control the interval of execution of its own array of subtasks, or else a subtask may be a vehicle controller and control the vehicle by coupling an input event to an output event. The vehicle controllers are the only tasks which directly control the vehicle and they do this by directing (i.e., coupling) the flow of data from an input event to an output event. Each controller has its associated event hierarchies. The event hierarchies are binary trees with the nodes being composed events and the leaves being device events. Using this architecture, a mission plan is represented by building trees of task and event objects. As the simple task and event objects perform their functions, the vehicle will physically execute the mission plan.

System Prototype

The NOSC prototype was a demonstration system for evaluating a distributable software architecture for plan execution, and not for developing a distributed operating system and interconnection network. For this reason, a single CPU multi-tasking system was implemented.

The approach was to first implement the software in the lab and execute plans under simulation. A personal computer was used for software development. After the system was lab tested, the EAVE-West vehicle console was modified to interface with the plan execution system. The plan execution system was then tested at a test pool.

The next step was to embed the plan execution system in the testbed. An additional computer was installed aboard the vehicle, and the vehicle hardware was modified to incorporate plan execution commands and communication with the new computer. At this point, the embedded system was tested in the lab under simulated conditions. The vehicle was then taken to a NOSC pier and tested in San Diego Bay.

Results

The prototype plan execution system has been implemented on the EAVE-West submersible and demonstrated. Simple maneuvers were performed off of a NOSC pier using a "Mission Planning Assistant" for vehicle trajectory input. This first in-bay exercise demonstrated the concept of using the proposed software architecture for general-purpose, stand-alone plan execution. Mission scenarios are now being developed.

Once installed, the plan execution system extended the capability of the testbed by providing the ability to specify and execute operator defined plans, thus demonstrating that vehicle development can progress along a continuum of capability. By increasing onboard processing, supervisory controlled vehicles can be made autonomous. Further extension along this continuum of capability is ensured, since the software is a distributed structure designed for multicomputing hardware.

Recommendations

Based on the successful experience with this project, the system should be implemented on a multicomputer. An accompanying high-level plan representation language should be developed to effectively use the system for more sophisticated scenarios. Finally, an "off-line" machine planner should be designed and implemented using that high-level language for the development of a responsive, knowledge-based, autonomous undersea vehicle system.

Acknowledgement

The Technical Director, Naval Ocean Systems Center, made this project possible by providing continued support via the Independent Exploratory Development Program (FY85, FY86, FY87).

References

[Doeling and Harding 87] Undersea Teleoperators and Intelligent Autonomous Vehicles, E. T. Harding and N. Doelling eds, MIT Sea Grant College Program, MITSG 87-1 (1987).

[Yoerger and Sheridan 83] "Supervisory Control Improves Performance for Underwater Telemanipulators", D. Yoerger and T. Sheridan, Marine Technology Society Meeting on Remotely Operated Vehicles, San Diego, CA, March 1983.

[Booch 83] Software Engineering With Ada, G. Booch, Benjamin/Cummings Publishing (1983).

[Booch 86] "Object-Oriented Development", G. Booch, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986.

[Harmon and Solorzano 83] "Information Processing Architecture for an Autonomous Robot System", S. Y. Harmon and M. R. Solorzano, Proc. of the Oakland Conference on Artificial Intelligence, Rochchester, MI, 26-27 April 1983.

[Aviles et al 85] "An Architecture for the Coordination and Control of Complex Robotic Subsystems", W. A. Aviles, S. Y. Harmon, D. W. Gage, G. L. Bianchini, 1985 Conf. on Intelligent Systems and Machines, Rochester, MI, April 1985.

[Harmon 81] "Autonomous Free Swimming Submersible: A Demonstration of Autonomous Robotics Technology", S. Y. Harmon, October 1981, (unpublished).

[Durham and Shirley 1982] "A Multi-Tasking Real Time Executive for the NOSC AFSS: An Introduction to Command Usage", J. T. Durham and R. W. Shirley, NOSC Memo 943/81-86, August 1982.

[Uhr 84] Algorithm-Structured Computer Arrays and Networks, L. Uhr, Academic Press (1984).

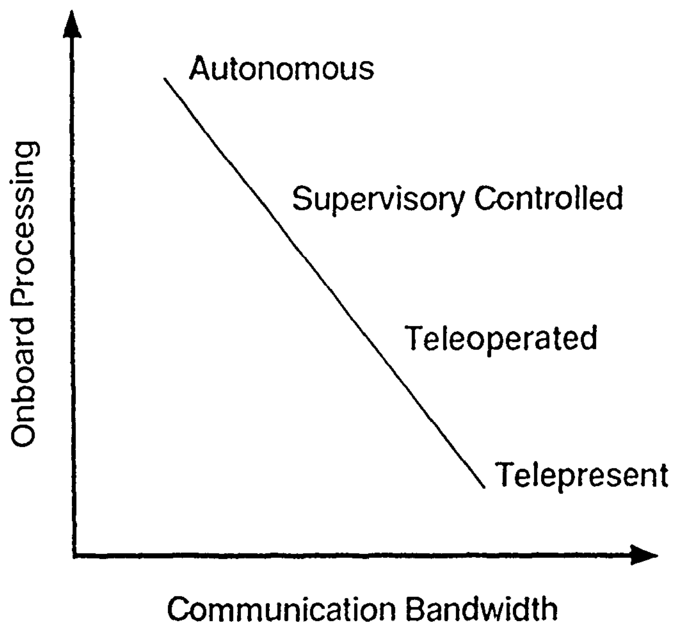

Figure 1. Continuum of ROV capability

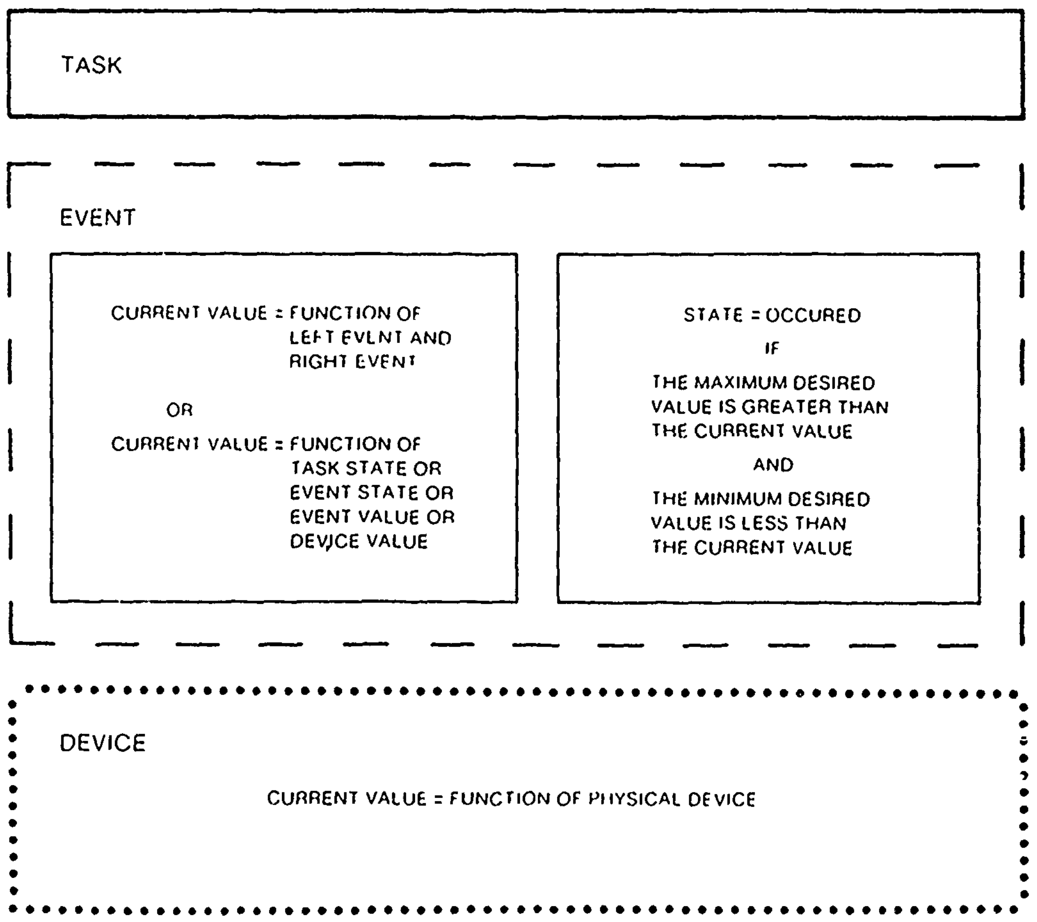
TASK
EVENT
CURRENT VALUE = FUNCTION OF LEFT EVENT AND RIGHT EVENT
OR
CURRENT VALUE = FUNCTION OF TASK STATE OR EVENT STATE OR EVENT VALUE OR DEVICE VALUE
STATE = OCCURED
IF
THE MAXIMUM DESIRED VALUE IS GREATER THAN THE CURRENT VALUE
AND
THE MINIMUM DESIRED VALUE IS LESS THAN THE CURRENT VALUE
DEVICE
CURRENT VALUE = FUNCTION OF PHYSICAL DEVICE

Figure 2 Layered plan execution primitives

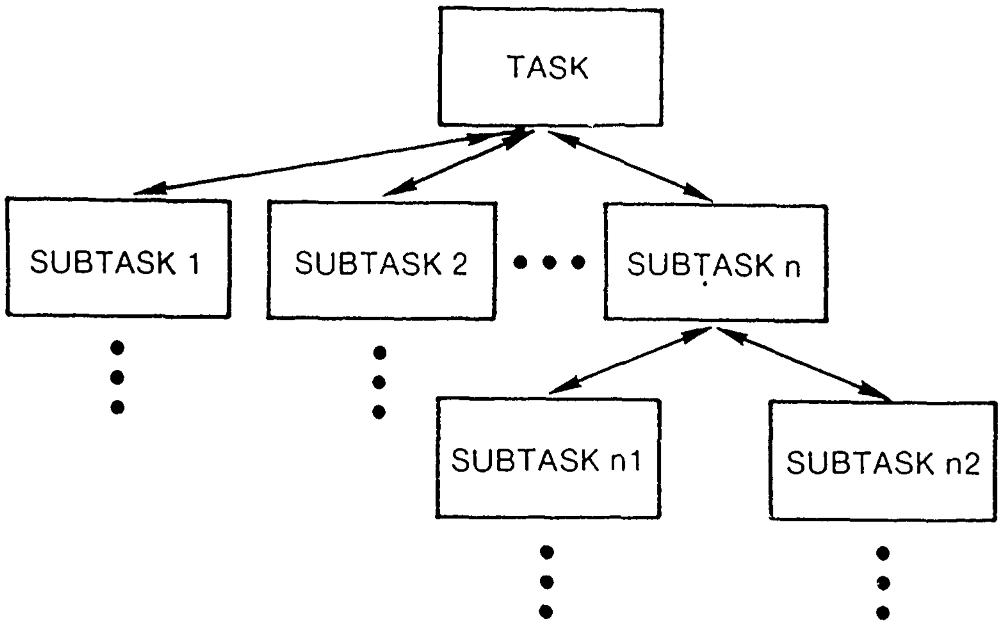

Figure 3 A control hierarchy

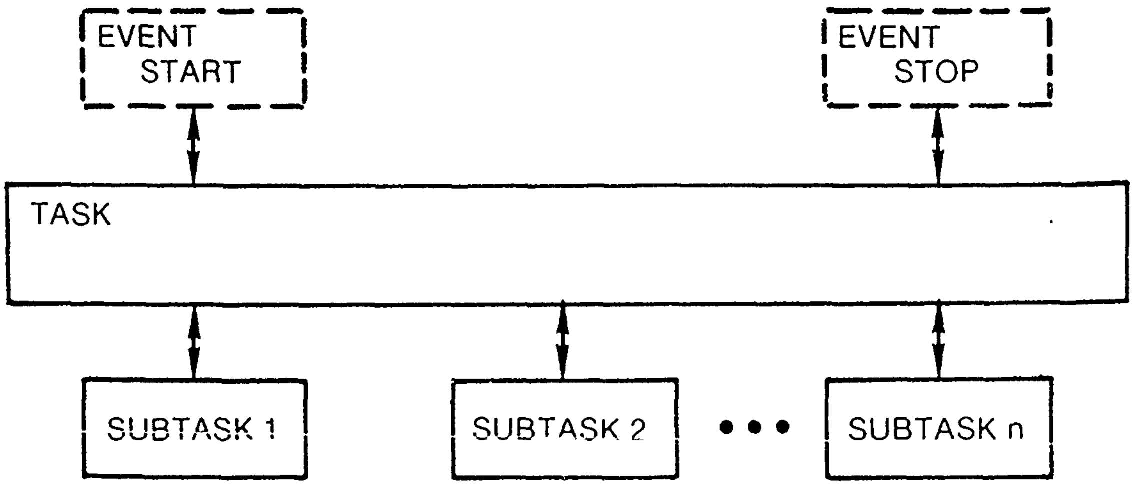

Figure 4 Control controller

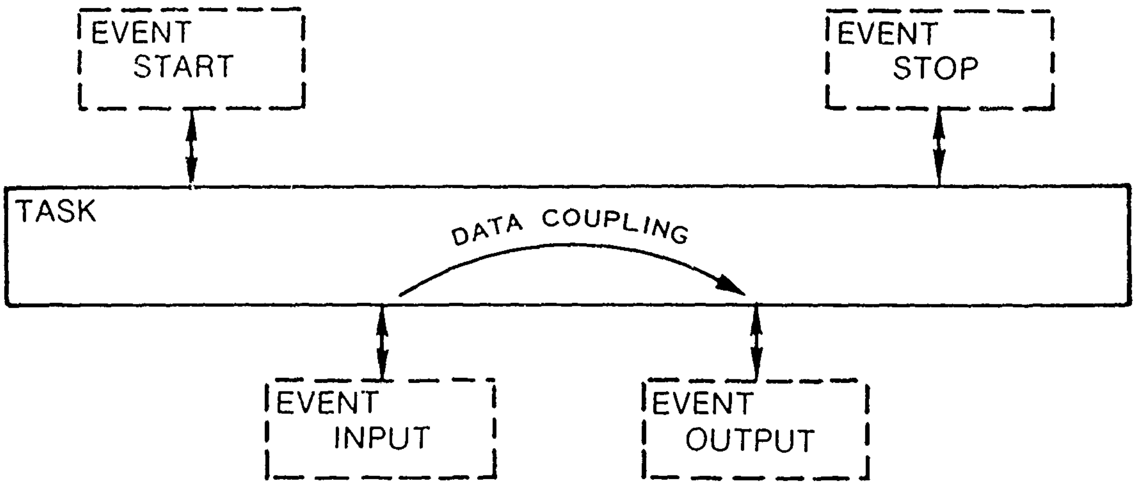

Figure 5 Vehicle controller

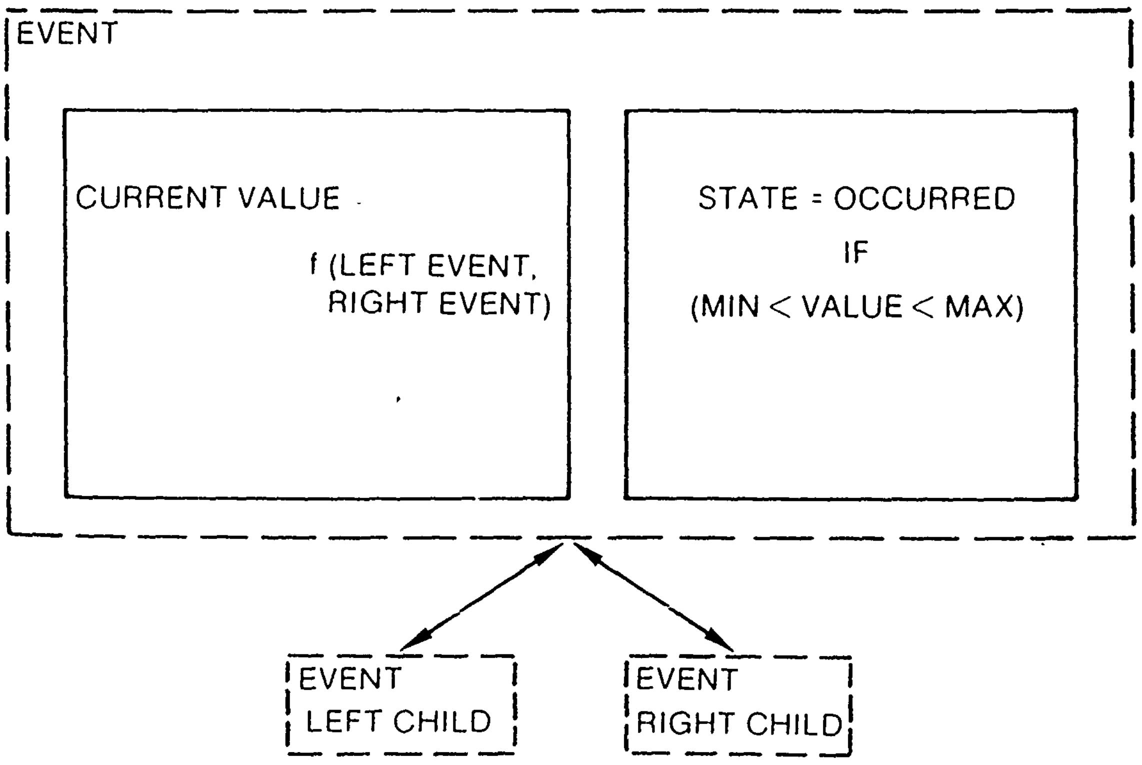

Figure 6 Composed event

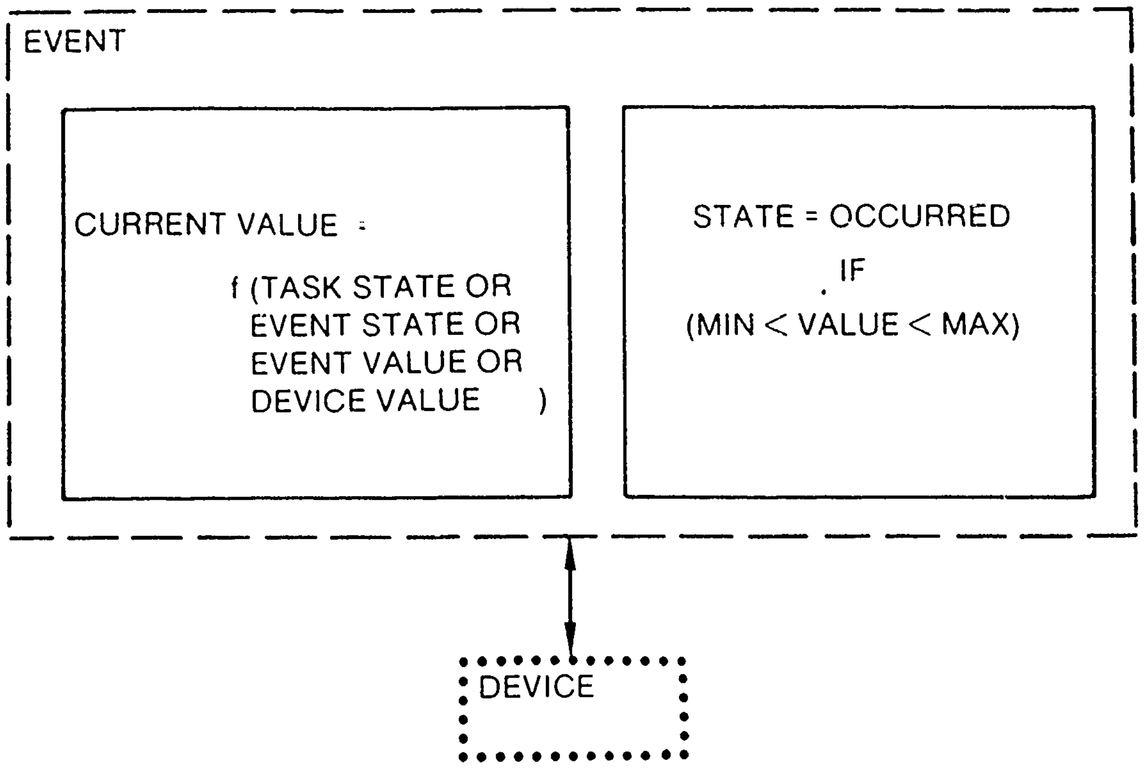

Figure 7 Device event